Text Clustering for Peer-to-Peer Networks with Probabilistic Guarantees

Odysseas Papapetrou[1], Wolf Siberski[1], and Norbert Fuhr[2]

[1] L3S Research Center {papapetrou,siberski}@l3s.de
[2] Universität Duisburg-Essen norbert.fuhr@uni-due.de

Abstract. Text clustering is an established technique for improving quality in information retrieval, for both centralized and distributed environments. However, for highly distributed environments, such as peer-to-peer networks, current clustering algorithms fail to scale. Our algorithm for peer-to-peer clustering achieves high scalability by using a probabilistic approach for assigning documents to clusters. It enables a peer to compare each of its documents only with very few selected clusters, without significant loss of clustering quality. The algorithm offers probabilistic guarantees for the correctness of each document assignment to a cluster. Extensive experimental evaluation with up to 100000 peers and 1 million documents demonstrates the scalability and effectiveness of the algorithm.

1 Introduction

Text clustering is widely employed for automatically structuring large document collections and enabling cluster-based information browsing, which alleviates the problem of information overflow. It is especially useful on large-scale distributed environments such as distributed digital libraries [8] and peer-to-peer (P2P) information management systems [2], since these environments operate on significantly larger document collections. Existing P2P systems also employ text clustering to enhance information retrieval efficiency and effectiveness [11, 17]. However, these systems do not address the problem of efficient *distributed* clustering computation; they assume that clustering is performed on a dedicated node, and are therefore not scalable. Existing distributed and P2P clustering approaches, such as [3, 5, 7, 9], are also limited to a small number of nodes, or to low dimensional data. Hence, a distributed clustering approach that scales to large networks and large text collections is required.
In this work we focus on text clustering for large P2P networks. We are particularly interested in systems in which the content distribution is imposed by the nature of the system, e.g., P2P desktop sharing systems [2]. We require a P2P clustering algorithm which can cluster such distributed collections effectively and efficiently, without overloading any of the participating peers, and without requiring central coordination.
A key factor to reduce network traffic for distributed clustering in these systems is to reduce the number of required comparisons between documents and clusters. Our approach achieves this by applying probabilistic pruning: Instead of considering all clusters for comparison with each document, only a few most

© Springer 2010

relevant ones are taken into account. We apply this core idea to K-Means, one of the frequently used text clustering algorithms. The proposed algorithm, called Probabilistic Clustering for P2P (PCP2P), reduces the number of required comparisons by an order of magnitude, with negligible influence on clustering quality.

In the following section, we present current distributed and P2P clustering algorithms, and explain why they are not applicable for P2P text clustering. In Section 3 we introduce PCP2P, an efficient and effective P2P text clustering algorithm. We present the probabilistic analysis for PCP2P in Section 4, and show how it is parameterized to achieve a desired correctness probability. In Section 5 we verify scalability and quality of PCP2P in large experimental setups, with up to 100000 peers and 1 million documents, using real and synthetic data.

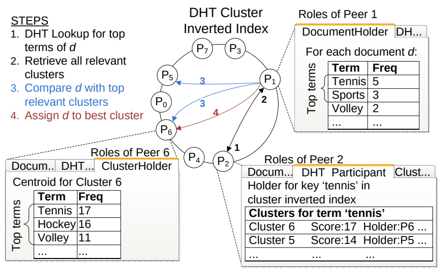
2 Related Work

Distributed Hash Tables. PCP2P relies on a Distributed Hash Table (DHT) infrastructure. DHTs provide efficient hash table capabilities in a P2P environment by arranging peers in the network according to a specific graph structure, usually a hypercube. DHTs offer the same functionality as their standard hash table counterparts, a `put(key,value)` method, which associates `value` with `key`, and a `get(key)` method, which returns the value associated with `key`. Our implementation uses Chord [16], but any other DHT could be used as well.

Distributed Clustering. Several algorithms for parallelizing K-Means have been proposed, e.g., [4, 6]. These algorithms focus on harnessing the power of multiple nodes to speed up the clustering of large datasets. They assume a controlled network or a shared memory architecture, and therefore are not applicable for P2P, where these assumptions do not apply.

Eisenhardt et al. [5] proposed one of the first P2P clustering algorithms. The algorithm distributes K-Means computation by broadcasting the centroid information to all peers. Due to this centroid broadcasting, it cannot scale to large networks. Hsiao and King [9] avoid broadcasting by employing a DHT. Clusters are indexed in the DHT using manually selected terms. This requires extensive human interaction, and the algorithm cannot adapt to new topics.

Datta et al. [3] proposed LSP2P and USP2P, two P2P approximations of K-Means. LSP2P uses gossiping to distribute the centroids. In an evaluation with 10-dimensional data, LSP2P achieves an average misclassification error of less than 3%. However, as we show in Section 5, LSP2P fails for text clustering, because it is based on the assumption that data is uniformly distributed among the peers, i.e., each peer has at least some documents from each cluster. This assumption clearly does not hold for text collections in P2P networks. The second algorithm, USP2P, uses sampling to provide probabilistic guarantees. However, the probabilistic guarantees are also based on the assumption that data is uniformly distributed among the peers. Also, USP2P requires a coordinating peer which gets easily overloaded, since it is responsible for exchanging centroids with a significant number of peers, for sampling, e.g., 500 peers out of 5500 peers.


STEPS
1. DHT Lookup for top terms of d
2. Retrieve all relevant clusters
3. Compare d with top relevant clusters
4. Assign d to best cluster
DHT Cluster Inverted Index
P7
P3
P5
P1
P0
P6
P4
P2
1
2
3
3
4
Roles of Peer 1
DocumentHolder
DH...
For each document d:
Top terms
Term Freq
Tennis 5
Sports 3
Volley 2
Roles of Peer 6
Docum..
DHT...
ClusterHolder
Centroid for Cluster 6
Top terms
Term Freq
Tennis 17
Hockey 16
Volley 11
Roles of Peer 2
Docum...
DHT Participant
Clust...
Holder for key 'tennis' in cluster inverted index
Clusters for term 'tennis'
Cluster 6 Score:17 Holder:P6 ...
Cluster 5 Score:14 Holder:P5 ...

Algorithm 1 PCP2P: Clustering the documents

```
1: for Document d in my documents do
   PRESELECTION STEP:
2:     CandClusters ← CandidateClustersFromDHT()
   FULL COMPARISON STEP:
3:     RemainingClusters ← FilterOut(candClusters)
4:     for Cluster c in RemainingClusters do
5:         Send d.termVector() to ClusterHolder(c)
6:         Sim[c] ← Retrieve similarity(d,c)
7:     end for
8:     Assign(d, cluster with max Sim)
9: end for
```

Fig. 1. PCP2P: a. System architecture, b. Algorithm for document assignment

Hammouda et al. [7] use a hierarchical topology for the coordination of K-Means computation. Clustering starts at the lowest level, and the local solutions are aggregated until the root peer of the hierarchy is reached. This algorithm has the disadvantage that clustering quality decreases noticeably for each aggregation level, because of the random grouping of peers at each level. Therefore, quality decreases significantly for large networks. The authors report a quality of less than 20% of the quality of K-Means already for 65 nodes.

3 PCP2P: Probabilistic Clustering for P2P

In PCP2P, a peer has up to three different roles (Fig. 1.a.). First, it serves as document holder, i.e., it keeps its own document collection, and it assigns its documents to clusters. Second, it participates in the underlying DHT by holding part of the distributed index. Third, a peer can be a *cluster holder*, i.e., maintain the centroid and document assignments for one cluster.

PCP2P consists of two parallel activities, *cluster indexing* and *document assignment*. Both activities are repeated periodically to compensate churn, and to maintain an up-to-date clustering solution. *Cluster indexing* is performed by the cluster holders. In regular intervals, these peers create compact cluster summaries and index them in the underlying DHT, using the most frequent cluster terms as keys. We describe this activity in Section 3.1. The second activity, *document assignment*, consists of two steps, preselection and full comparison. In the *preselection step*, the peer holding d retrieves selected cluster summaries from the DHT index, to identify the most relevant clusters (Fig. 1.b, Line 2). Preselection already filters out most of the clusters. In the *full comparison step*, the peer computes similarity score estimates for d using the retrieved cluster summaries. Clusters with low similarity estimates are filtered out (Line 3, see Section 3.2 for details), and the document is sent to the few remaining cluster holders for full similarity computation (Lines 4-7). Finally, d is assigned to the cluster with highest similarity (Line 8). This two-stage filtering algorithm reduces drastically the number of full comparisons (usually less than five comparisons per document, independent of the number of clusters). At the same time, it provides probabilistic guarantees that the resulting clustering solution exhibits nearly the same quality as centralized clustering (Section 4).

3.1 Indexing of Cluster Summaries

Cluster holders are responsible for indexing summaries of the clusters in the DHT. Particularly, each cluster holder periodically recomputes its cluster centroid, using the documents assigned to the cluster at the time. It also recomputes a *cluster summary* and publishes it to the DHT index, using selected cluster terms as keys. As we explain later, this enables peers to identify relevant clusters for their documents efficiently. For this identification, it is sufficient to consider the most frequent terms of a cluster c as keys, i.e., all terms t with $TF(t, c) \geq CluTF_{min}(c)$, where $CluTF_{min}(c)$ denotes the frequency threshold for c. We use $TopTerms(c)$ to denote the set of these terms. Note that $TopTerms(c)$ does not include stopwords; these are already removed when building the document vectors. For the rest of this section we assume that $CluTF_{min}(c)$ is given. Section 4 shows how a value for this threshold can be derived that satisfies the desired probabilistic guarantees.
The cluster summary includes (1) all cluster terms in $TopTerms(c)$ and their corresponding TF values, (2) $CluTF_{min}(c)$, and (3) the sum of all term frequencies (the L1 norm), cluster length (the L2 norm), and dictionary size.
Load Balancing. To avoid overloading, each cluster holder selects random peers to serve as *helper cluster holders*, and replicates the cluster centroid to them. Their IP addresses are also included in the cluster summaries, so that peers can randomly choose a replica without going through the cluster holder. Communication between the master and helper cluster holders only occurs for updating the centroids, by exchanging the respective local centroids as in [4]. Since only one centroid needs to be transferred per helper cluster holder and only a small number of peers is involved, load balancing does not affect scalability.

3.2 Document Assignment to Clusters

Each peer is responsible of clustering its documents periodically. Clustering of a document consists of two steps: (a) the preselection step, where the most promising clusters for the document are detected, and, (b) the full comparison step, where the document is fully compared with the most promising clusters and assigned to the best one.
Preselection step. Consider a peer p which wants to cluster a document d. Let $TopTerms(d)$ denote all terms in d with $TF(t, d) \geq DocTF_{min}(d)$, where $DocTF_{min}(d)$ denotes a frequency threshold for d (we explain how $DocTF_{min}$ is derived in Section 4). For each term t in $TopTerms(d)$, peer p performs a DHT lookup and finds the peer that holds the cluster summaries for t (Fig. 1.a, Step 1). It then contacts that peer directly to retrieve all summaries published using t as a key (Step 2). To avoid duplicate retrieval of summaries, p executes these requests sequentially, and includes in each request the cluster ids of all summaries already retrieved. We refer to the list of all retrieved summaries as the *preselection list*, denoted with $\mathcal{C}_{pre}$. The summary of the optimal cluster for d is included in $\mathcal{C}_{pre}$ with high probability, as shown in Section 4.

Full comparison step. After constructing $\mathcal{C}_{pre}$, peer p progressively filters out the clusters not appropriate for the document at hand, using one of two alternative filtering strategies, as follows. Using the retrieved cluster summaries, p estimates the cosine similarities for all clusters in $\mathcal{C}_{pre}$. For the cluster with the highest similarity estimate, p sends the compressed term vector of d to the respective cluster holder for a full cosine similarity, and retrieves the similarity score (Fig. 1.a, Step 3). Based on this score and the employed filtering strategy, more clusters are filtered out. The process is repeated until $\mathcal{C}_{pre}$ is empty. Finally, p assigns d to the cluster with the highest similarity score, and notifies the respective cluster holder (Step 4).
Filtering Strategies. We propose two different strategies to filter out clusters from $\mathcal{C}_{pre}$, (a) *conservative*, and (b) *Zipf-based filtering*. Both strategies employ the information contained in the cluster summaries to estimate the cosine similarity between the document and each candidate cluster. Let $t_1, t_2, \ldots, t_n$ denote the terms of d sorted *descending* by their frequency, i.e., $TF(t_i, d) \geq TF(t_j, d)$ for all $j > i$. Cosine similarity is estimated as follows.

$$ECos(d,c) = \sum_{i=1}^{n} \frac{TF(t_i,d) \times f(t_i,c)}{|d| \times |c|} \tag{1}$$

$|d|$ and $|c|$ denote the L2-Norm of the document and cluster. The function $f(t_i, c)$ denotes an estimation for $TF(t_i, c)$, which is specific to the filtering strategy. We will address this function in detail in the next paragraphs.
Having estimated the cosine similarities for all clusters in $\mathcal{C}_{pre}$, PCP2P proceeds as follows. Let $c_{\overline{max}}$ denote the cluster in $\mathcal{C}_{pre}$ with the maximum estimated similarity. Peer p removes $c_{\overline{max}}$ from the list and sends the compressed term vector of d to the cluster holder of $c_{\overline{max}}$, for cosine comparison. After retrieving the real cosine value $Cos(d, c_{\overline{max}})$, it removes from $\mathcal{C}_{pre}$ all clusters c with $ECos(d, c) < Cos(d, c_{\overline{max}})$. This process is repeated, until $\mathcal{C}_{pre}$ is empty. Finally, d is assigned to the cluster with the highest cosine similarity.
The key distinction between the two filtering strategies is the way they compute $ECos(d, c)$, and, in particular, their definition of $f(t_i, c)$.
Conservative filtering. The conservative strategy computes an upper bound for cosine similarity. For the terms included in the cluster's summary, conservative strategy uses the actual cluster frequency, included in the summary. For all other terms, it progressively computes an upper bound for the term frequency in the cluster. Formally, $f(t_i, c)$ is defined as follows.

$$f(t_i,c) = \begin{cases} TF(t_i,c) & \text{if } t_i \in TopTerms(c) \\ min(CluTF_{min}(c) - 1, SC - ST - SE) & \text{otherwise} \end{cases}$$

where SC is the sum of cluster frequencies for all terms included in the cluster, and ST is the sum of cluster frequencies for all terms included in $TopTerms(c)$. SE holds the sum of all term frequencies estimated up to now by the algorithm for c. By definition, the conservative strategy never underestimates the cosine similarity value. Therefore, this strategy always detects the best cluster.
Zipf-based filtering. A more accurate similarity estimation can be derived based on the assumption that term frequencies in the cluster follow a Zipf distribution. Recall that document terms $t_1, t_2, \ldots, t_n$ are ordered descending on

their frequency. We use this order to estimate the rank of terms not included in the cluster summary. Zipf-based filtering defines $f(t_i, c)$ as follows:

$$f(t_i, c) = \begin{cases} TF(t_i, c) & \text{if } t_i \in TopTerms(c) \\ min(SC/(r^s \times \sum_{k=1}^{DT} 1/k^s), SC - ST - SE) & \text{otherwise} \end{cases}$$

SC, ST, and SE are defined as in the conservative strategy. DT denotes the number of distinct terms in c, and with r we represent the estimated rank for the missing term. $SC/(r^s \times \sum_{k=1}^{DT} 1/k^s)$ gives the expected term frequency of t_i in c, assuming that term frequencies follow a Zipf distribution with exponent s. Ranks of missing terms are estimated as follows: the i-th document term that is not included in the $TopTerms(c)$ is assumed to exist in the cluster centroid, with rank $r = |TopTerms(c)| + i$.

3.3 Cost analysis

We express cost in number of messages and transfer volume. For a cluster c, the cost of indexing the cluster summary (both in number of messages and transfer volume) is $Cost_{ind} = O(|TopTerms(c)| \times \log(n))$, where n is the number of peers. The cost of the preselection step for each document d is $Cost_{pre} = O(|TopTerms(d)| \times \log(n))$. The full comparison step incurs a cost of $Cost_{fcs} = O(|\mathcal{C}_{fcs}|)$, where $\mathcal{C}_{fcs}$ denotes the set of clusters fully compared with d.
The dominating cost is the one incurred for assigning documents to clusters, namely $Cost_{pre} + Cost_{fcs}$. Per document, this cost has the following properties: (a) it grows logarithmically with the number of peers, because DHT access cost grows logarithmically, and (b) it is independent of the size of the document collection. It also depends on $|\mathcal{C}_{fcs}|$. Our experimental evaluation (Section 5) shows that $|\mathcal{C}_{fcs}|$ is on average very small, and independent of the total number of clusters k. This means that PCP2P scales to networks of large sizes, and with large numbers of documents and clusters.

4 Probabilistic Analysis

In the previous section, we assumed that the optimal values for $CluTF_{min}(c)$ and $DocTF_{min}(d)$ are given. We now describe how PCP2P computes these values dynamically for each cluster and document to satisfy the desired clustering quality requirements. Due to space limitations, we only provide the final results of the analysis here. The reader can find the full proofs in [14].
Our analysis uses a probabilistic document generation model [13, 15]. Briefly, the model assumes that each document belongs to a topic $\mathcal{T}$, and each topic $\mathcal{T}_i$ is described by a term probability distribution ϕ_i (a language model). A document of length l that belongs to $\mathcal{T}_i$ is created by randomly selecting l terms with replacement from ϕ_i. The probability of selecting a term t is given by ϕ_i.
Notations. $Pr_{correct}$ denotes the desired correctness probability, i.e., the probability of each document to be assigned to the correct cluster. We use $\mathcal{C}_{sol} := \{c_1, \ldots, c_k\}$ to denote a snapshot of clusters on an ongoing clustering.

Each cluster $c_i \in \mathcal{C}_{sol}$ follows the language model ϕ_i. We use $t_1[\phi_i], \dots, t_n[\phi_i]$ to denote the terms of ϕ_i sorted by descending probabilities. Also, $TopDistr(\alpha, \phi_i)$ denotes the set of α terms with highest probability in ϕ_i, i.e., $t_1[\phi_i]$, …, $t_\alpha[\phi_i]$. For the case of conservative filtering, a document d is assigned correctly to its optimal cluster, denoted with c_{opt}, if c_{opt} is detected in the preselection step and included in $\mathcal{C}_{pre}$. The probability that c_{opt} is included in $\mathcal{C}_{pre}$ is denoted with Pr_{pre}. Clearly, by setting $Pr_{pre} = Pr_{correct}$ we satisfy the desired correctness probability. The purpose of the analysis is to find the values of $CluTF_{min}$ and $DocTF_{min}$ that satisfy Pr_{pre}.
PCP2P computes these values automatically, as follows. c_{opt} is retrieved in the preselection step if there exists at least one term $t \in TopDistr(\alpha, \phi_i)$ with $TF(t, c_{opt}) \geq CluTF_{min}$, and at the same time $TF(t, d) \geq DocTF_{min}$. Let Pr_{find} denote the probability that each of the terms from $TopDistr(\alpha, \phi_i)$ has a frequency in d of at least $DocTF_{min}$. With Pr_{ind} we denote the probability that each term from $TopDistr(\alpha, \phi_i)$ has a frequency in c_{opt} of at least $CluTF_{min}$. With Eqn. 4, peers compute the proper values for Pr_{find} and Pr_{ind}, such that clustering succeeds with probability Pr_{pre}. Given these probabilities, each peer computes the proper values for $DocTF_{min}$ and $CluTF_{min}$ per document and cluster respectively, according to Eqns. 2 and 3.
The probabilistic guarantees are based on the following theorem.

Theorem 1. *Given a document d which follows language model ϕ_i. The expected frequency of term t in d according to ϕ_i is denoted with $\hat{TF}(t, d)$. For any term t with $\hat{TF}(t, d) > DocTF_{min}$, the probability of the actual term frequency $TF(t, d)$ exceeding $DocTF_{min}$ is at least $1 - \exp(-\hat{TF}(t, d) \times (1 - DocTF_{min}/\hat{TF}(t, d))^2/2)$. Furthermore, with a probability Pr_{find} the frequency of term t in d is at least $\hat{TF}(t, d) - \sqrt{2 \times \hat{TF}(t, d) \times \log\left(\frac{1}{1 - Pr_{find}}\right)}$.*

Sketch. The proof uses the lower-tail Chernoff bound ([12], p. 72) to compute a lower bound for the term frequency of a term t in d according to ϕ_i. In particular, we model the generation of d as independent Poisson trials, using the term probabilities of ϕ_i, as is standard in language generation models. Then, we apply Chernoff bounds to find $Pr[TF(t, c_i) < DocTF_{min}]$. Finally:

$$\begin{aligned} Pr[TF(t, d) \geq DocTF_{min}] &= 1 - Pr[TF(t, c_i) < DocTF_{min}] \\ &\geq 1 - \exp(-\hat{TF}(t, d) \times (1 - DocTF_{min}/\hat{TF}(t, d))^2/2) \quad (2) \end{aligned}$$

The second part of the theorem is derived by solving Eqn. 2 for $DocTF_{min}$. □
With a similar theorem (see [14]), we can find a probabilistic lower bound for the frequency of a term t in c_i:

$$CluTF_{min} = \hat{TF}(t, c_i) - \sqrt{2 \times \hat{TF}(t, c_i) \times \log\left(\frac{1}{1 - Pr_{ind}}\right)} \quad (3)$$

To compute the expected term frequencies for $t_x[\phi_i]$, i.e., the x'th most frequent term in ϕ_i, peers use the Zipf distribution (validated, for example, in [1] for text).

Using Eqn. 3, the cluster holder of c_i selects $CluTF_{min}$ such that $Pr[TF(t_\alpha[\phi_i], c_i) \geq CluTF_{min}] \geq Pr_{ind}$. The probability that both d and c_{opt} have a common term with corresponding frequencies at least $DocTF_{min}$ and $CluTF_{min}$, is at least:

$$Pr_{pre} \geq 1 - \prod_{j=1}^{\alpha} (1 - Pr[TF(t_j, d) \geq DocTF_{min}] \times Pr_{ind}) \tag{4}$$

Using Eqn. 4, peers set the value of $DocTF_{min}$ per document, s.t. $Pr_{pre} \geq Pr_{correct}$.
Algorithm Configuration. As in standard K-Means, the number of clusters k can be freely chosen. In addition, PCP2P allows to set the desired correctness probability $Pr_{correct}$. All other parameters for satisfying the required probabilistic guarantees are derived using the results of our analysis.
First, a few sampled documents are collected from the network and are used to estimate the Zipf distribution skew. The algorithm then computes the remaining parameters. By default, α is set to 5, and Pr_{ind} and Pr_{pre} are set to $Pr_{correct}$. As shown earlier, these probability values satisfy the desired correctness probability for conservative filtering. Each peer then computes the proper $DocTF_{min}$ and $CluTF_{min}$ per document and cluster which satisfy these probabilities. With respect to Zipf-based filtering, we offer probabilistic guarantees only for the preselection step. The above values also satisfy the probabilistic guarantees for the preselection step of Zipf-based filtering.

5 Experimental Evaluation

The purpose of the experiments was to evaluate PCP2P with respect to effectiveness, efficiency, and scalability. Effectiveness was evaluated based on a human-generated classification of documents, using the two standard quality measures, entropy and purity. Additionally, we measured how well PCP2P approximates K-Means. With respect to efficiency, we measured number of messages, transfer volume, and document-cluster comparisons, for different correctness probabilities and collection characteristics. Finally, we examined scalability of PCP2P by varying the network and collection size, and the number of clusters. In the following we report average results after 10 repetitions of each experiment. Unless otherwise mentioned, we report results for 50 clusters.
As a real-world dataset, we have used the REUTERS Corpus Volume 1 (RCV1) [10]. We chose RCV1 because it is the largest collection with a classification of articles, which is necessary for evaluation of clustering. To be able to apply standard quality measures, we restricted RCV1 to all articles which belonged to exactly one class (approx. 140000 articles). To systematically examine the effect of the collection's characteristics on the algorithm, and to evaluate it with a significantly larger dataset, we also used synthetic document collections (SYNTH) with a size of 1.4 million documents each. These collections were created according to the well-accepted Probabilistic Topic Model [15] from 200 composite language models, with different term distribution skews.
We compared PCP2P with two other P2P clustering algorithms: (a) LSP2P [3], the state-of-the-art for P2P clustering, and, (b) DKMeans, a P2P implementation of K-Means. DKMeans works like PCP2P, but without preselection and

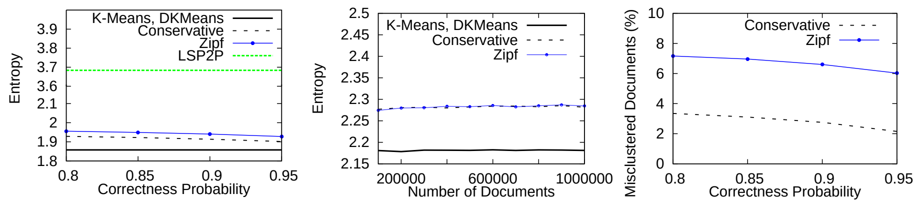


Fig. 2. Quality: a. Entropy, b. Entropy for different dataset sizes, c. Approximation

filtering, i.e., each compressed document vector is sent to all cluster holders for comparison purposes before assigned to a cluster. DKMeans was included in our experiments because it accurately simulates K-Means, i.e., it produces exactly the same results in a distributed fashion.

5.1 Evaluation Results

Clustering Quality. Figure 2.a plots entropy of all algorithms for the RCV1 dataset (lower entropy denotes better clustering quality). The X-axis is only relevant to the two PCP2P approaches, but we also include K-Means and LSP2P results for comparison. For clarity, the Y-axis is discontinuous. For the reported experiments we have used a network of 10000 peers, with 20% churn. Note that network size has no effect on the quality of PCP2P and DKMeans.
We see that both PCP2P filtering strategies achieve nearly the same clustering quality as K-Means. On the other hand, LSP2P, the current state-of-the-art in P2P clustering, converges to a significantly worse clustering solution, even for a moderate networks of 10000 peers. LSP2P fails because it is based on the assumption that each peer has at least some documents from each cluster, which is unsatisfiable with respect to text clustering [3]. Concerning the PCP2P variants, conservative filtering yields the best (lowest) entropy and the best (highest) purity, as expected. Zipf-based filtering shows comparable results. The same outcome is observed with respect to purity: K-Means has a purity of 0.645, and the maximum relative difference between PCP2P and K-Means is less than 1%. LSP2P has an average purity of 0.29, which shows insufficient clustering quality. Since LSP2P fails to produce a clustering solution of comparable quality with PCP2P and K-Means, we do not include it in our further experiments.
To evaluate the effect of document collection size on clustering quality, we repeated the experiment with the SYNTH collection, varying the number of documents between 100000 and 1 million. The SYNTH collection was generated as explained earlier, with a term distribution skew of 1.0 [18]. Figure 2.b plots entropy in correlation to the number of documents, using correctness probability $Pr_{correct} = 0.9$. We see that for PCP2P, entropy remains very close to the entropy achieved by K-Means, even for the largest collection with one million

[3] Note that we reproduced the good results of LSP2P on a smaller network with a 10-dimensional dataset, as reported in [3].

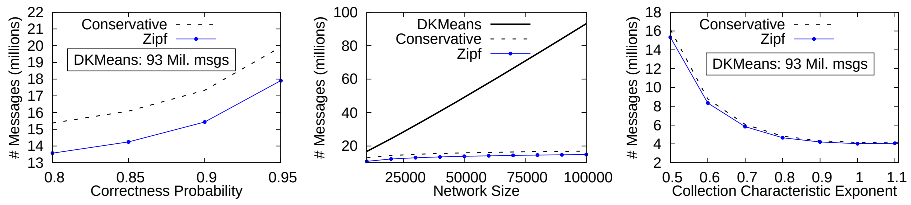


Fig. 3. Efficiency for varying: a. $Pr_{correct}$, b. Network size, c. Term Distribution Skew

	Messages		Transfer vol.(Mb)		Comparisons		Entropy	
Clusters	25	100	25	100	25	100	25	100
DKMeans	4.66E7	1.86E8	2176	8706	25E5	100E5	2.03	1.61
Conservative	1.45E7	1.70E7	995	2490	568078	1704332	2.08	1.66
Zipf	1.34E7	1.36E7	603	1228	10673	31907	2.12	1.69

Table 1. Cost and Quality of PCP2P and DKMeans for varying number of clusters.

documents. The same applies for purity. This confirms that quality of PCP2P is not affected by collection size. We also see that for collections with high term distribution skews (1.0 in this experiment, compared to 0.55 for the RCV1 collection), Zipf-based filtering is nearly as effective as conservative filtering.

Approximation quality. In addition to the standard quality measures, we also counted the number of documents that PCP2P assigned to a different cluster than K-Means after each clustering round. Fig. 2.c plots the percentage of misclustered documents for different values of $Pr_{correct}$. As expected, conservative yields the best approximation, with less than 4% misclusterings even for $Pr_{correct} =$ 0.8. Zipf-based PCP2P is also very accurate. We also see that the actual number of misclustered documents for PCP2P is always better than the probabilistic guarantees, since guarantees refer to upper bounds for number of errors.

Efficiency. We used PCP2P and DKMeans to cluster the RCV1 collection in a network of 100000 peers, with 20% churn. We did not include LSP2P here, since it fails with respect to quality, as already shown. Figure 3.a shows the number of messages required to perform one clustering iteration in correlation to $Pr_{correct}$. The plot also includes DKMeans cost as reference. We see that the two PCP2P variants generate significantly fewer messages than DKMeans. Also, as expected, Zipf-based filtering is more efficient than conservative filtering. For $Pr_{correct} < 0.9$, Zipf-based filtering requires an order of magnitude less messages than DKMeans, and conservative filtering requires less than 20% of the messages. The reason for the significantly better performance of PCP2P compared to DKMeans is that it reduces the number of document-cluster comparisons. Conservative filtering requires less than 20% of the comparisons of DKMeans in all setups, whereas Zipf requires less than 1%. Therefore, documents are sent over the network fewer times. Regarding transfer volume, conservative filtering requires less than 22% of the respective transfer volume of DKMeans, whereas Zipf-based filtering requires around 5%.

Scalability. We evaluated scalability of PCP2P with respect to network size, number of documents, and number of clusters. Figure 3.b shows the cost for different network sizes. The cost for PCP2P increases only logarithmically with network size, while cost for DKMeans increases linearly. This behavior is expected, because the only factor changing with network size for PCP2P is the DHT access cost, which grows logarithmically. The same behavior is observed with respect to transfer volume. We do not show quality measures, because they are independent of the network size. Our experimental results also confirmed that PCP2P scales linearly with collection size, as shown in Section 3.3. We repeated the clustering of RCV1 with 25 and 100 clusters, on a network of 100000 peers. Table 1 summarizes the results. Both PCP2P filtering strategies scale well with the number of clusters. In fact, network savings of PCP2P grow with the number of clusters, compared to DKMeans. Regarding entropy, the relative difference between PCP2P variants and DKMeans is stable. The same is observed regarding purity (not included in the table). Summarizing, PCP2P approximation quality is independent of the number of clusters, but PCP2P cost savings become even higher for a larger number of clusters.
Influence of Term Distribution Skew. PCP2P relies on the fact that term frequencies follow a Zipf distribution. Although it is accepted that document collections follow Zipf distribution, different document collections exhibit different distribution skews [1]. For example, the RCV1 collection used in our experiments has a skew of 0.55, while values reported in the literature for other text collections are around 1.0 [1, 18]. To evaluate the influence of the skew on PCP2P, we used SYNTH collections generated with different Zipf skew factors, between 0.5 and 1.1. We do not present details with respect to quality, because the quality of PCP2P was always high and unaffected by the skew factor.
Figure 3.c displays the execution costs in number of messages, for a varying distribution skew, and for $Pr_{correct} = 0.9$. To achieve the same quality level, PCP2P cost is significantly lower for higher skews. This behavior is expected: with higher skews PCP2P needs to perform fewer document-cluster comparisons for satisfying the probabilistic guarantees. For commonly reported skew values (around 1.0), the number of messages is reduced by an order of magnitude. But even for a skew as low as 0.5, the cost of both PCP2P variants is significantly lower than the cost of DKMeans.

6 Conclusions

We presented PCP2P, the first scalable P2P text clustering algorithm. PCP2P achieves a clustering quality comparable to standard K-Means, while reducing communication costs by an order of magnitude. We provided a probabilistic analysis for the correctness of the algorithm, and showed how PCP2P automatically adapts to satisfy the required probabilistic guarantees. Extensive experimental evaluation with real and synthetic data confirm the efficiency, effectiveness and scalability of the algorithm, and its appropriateness for text collections with a wide range of characteristics.

Our future work focuses on applying the core idea of PCP2P, i.e., probabilistic filtering, to other clustering algorithms, both for distributed and centralized settings. Furthermore, we work towards a P2P IR method based on clustering, similar to [11, 17], but now based on PCP2P, a truly distributed clustering infrastructure.

References

1. C. Blake. A comparison of document, sentence, and term event spaces. In *ACL*, 2006.
2. P. Cudré-Mauroux, S. Agarwal, and K. Aberer. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing*, 11(5), 2007.
3. S. Datta, C. R. Giannella, and H. Kargupta. Approximate distributed K-Means clustering over a peer-to-peer network. *IEEE TKDE*, 21(10):1372–1388, 2009.
4. I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Workshop on Large-Scale Parallel KDD Systems*, 1999.
5. M. Eisenhardt, W. Müller, and A. Henrich. Classifying documents by distributed P2P clustering. In *INFORMATIK*, 2003.
6. G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explor. Newsl.*, 2(2):34–38, 2000.
7. K. Hammouda and M. Kamel. HP2PC: Scalable hierarchically-distributed peer-to-peer clustering. In *SDM*, 2007.
8. B. Haslhofer and P. Knezević. The BRICKS digital library infrastructure. In *Semantic Digital Libraries*, pages 151–161. 2009.
9. H.-C. Hsiao and C.-T. King. Similarity discovery in structured P2P overlays. In *ICPP*, 2003.
10. D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: a new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.
11. J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *CIKM '03*, pages 199–206, New York, NY, USA, 2003. ACM.
12. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
13. C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: a probabilistic analysis. In *PODS*, 1998.
14. O. Papapetrou, W. Siberski, and N. Fuhr. Text clustering for P2P networks with probabilistic guarantees. Extended version, 2009. `http://www.l3s.de/~papapetrou/publications/pcp2p-ecir-ext.pdf`.
15. M. Steyvers and T. Griffiths. *Handbook of Latent Semantic Analysis*, chapter Probabilistic Topic Models, pages 427–448. Lawrence Erlbaum, 2007.
16. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.
17. J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *SIGIR*, 1999.
18. G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.